Software in the CARMA heterogeneous millimeter-wave array

Stephen L. Scott

Stephen L. Scott, "Software in the CARMA heterogeneous millimeter-wave array," Proc. SPIE 7019, Advanced Software and Control for Astronomy II, 701904 (14 July 2008); doi: 10.1117/12.788504

SPIE.

Event: SPIE Astronomical Telescopes + Instrumentation, 2008, Marseille, France

Software in the CARMA Heterogeneous Millimeter-wave Array

Stephen L. Scott*
California Institute of Technology, Owens Valley Radio Observatory,
PO Box 968, Big Pine, CA, USA 93513

ABSTRACT

The Combined Array for Research in Millimeter-wave Astronomy (CARMA) is a 15 element heterogeneous millimeter-wave array developed and operated by a university consortium that will be expanded to 23 elements in 2008. Commissioning began in August 2005 after completion of the relocation of antennas from the Owens Valley Radio Observatory (OVRO) and the Berkeley-Illinois-Maryland Association (BIMA) arrays to a new high site and initial scientific operations began in April 2006. The array operates in the 3-mm and 1-mm bands and has a maximum resolution of 0.15 arc seconds. Most of the software and computing infrastructure for the array is new, allowing modern technology to be introduced and to provide a common interface for the disparate antenna types. The new system is proving to be both easy to use for routine observations and yet capable enough for the development of new observing techniques by the experienced astronomer. Some of the details of the computing and software are described here, with emphasis on the control system.

Keywords: CARMA, millimeter-wave interferometry, radio astronomy, control systems, software

1. INTRODUCTION

The Combined Array for Research in Millimeter-wave Astronomy (CARMA) consolidates several decades of research in millimeter-wave interferometric technology and science into a more capable instrument at a superior site. The Owens Valley Radio Observatory (OVRO) brings six 10.4-m antennas and the Berkeley-Illinois-Maryland Association (BIMA) contributes nine 6.1-m antennas. The antennas were modified to use a common base that was compatible with a single transporter and station design. The new site, Cedar Flat, at an elevation of 2200 m (7200') in the Inyo Mountains, is about a 20 minutes drive from OVRO and is shown with the array in Figure 1. The receivers operate in the 3-mm and 1-mm bands with an instantaneous bandwidth of 4 to 8 GHz in each of two sidebands. They feed a 15 station correlator with three 500 MHz wide basebands which will be expanded to eight basebands in 2008. Maximum spatial resolution in the 2 km A array is 0.15 arcseconds.

In mid-2008 the Sunyaev-Zel'dovich Array (SZA) of eight 3.5-m antennas will move to Cedar Flat, bringing the number of antennas to 23. These antennas operate at 1-cm with 8 GHz bandwidth, all of which is processed by an eight station coarse resolution wideband correlator. In 2009 3-mm capability will be added to the SZA antennas and 1-cm will be added to the OVRO antennas. At this time all 23 antennas will be inter-operable, controlled by the same software, with their signals feeding either the 15 station correlator or the 8 stations correlator via an RF switchyard. With the addition of the SZA the consortium will consist of the California Institute of Technology, the University of California, Berkeley, the University of Illinois at Urbana-Champaign, the University of Maryland, and the University of Chicago. Further details of the CARMA system are found with a description of the first results in Bock, et al., 2006[1].

The computing infrastructure is divided between a proposal submission, data archive, and imaging pipeline at the National Center for Supercomputing Applications (NCSA) at the University of Illinois, and the realtime system at Cedar Flat. Many algorithms, much expertise, and some code were recycled from the previous arrays into CARMA, but the majority of the code base is the result of new development to give a modern and cohesive foundation. Requirements for the new system were developed by late 2002 and the staff ramped up to about 14 people (~9 FTEs) and design work started by mid-2003. Coding was begun in early 2004 and gathered momentum rapidly throughout the year. The system was sufficiently advanced for first fringes in August 2005, adding operational responsibilities to the computing staff.

* scott@ovro.caltech.edu; phone +1 760 938-2075 x110; http://www.mmarray.org

Advanced Software and Control for Astronomy II, edited by Alan Bridger, Nicole M. Radziwill
Proc. of SPIE Vol. 7019, 701904, (2008) · 0277-786X/08/$18 · doi: 10.1117/12.788504

Shared risk science began in April 2006, evolving into routine science by the beginning of 2007. Reductions in computing staff began in early 2005 leading to a current staffing level of about six FTEs for computing operations and development.

The array is run 24/7 by graduate students, postdocs, and faculty, usually from the partner universities, scheduled two at a time. A standard shift is one week, with occasional shifts of two weeks, with the change days for the observers scheduled to overlap in the middle of the shift. Maintaining continuity of information is a challenge, with several other mechanisms besides the overlap used to preserve knowledge of the current array conditions. A 'friend of the array' stationed at OVRO provides an initial point of contact for many queries, supplemented by web based documentation. The observers have lunch at OVRO in the valley every work day, which is immediately followed by a face-to-face meeting with senior operational and development staff that are stationed at OVRO. The observers select projects in realtime based on atmospheric conditions, the current configuration, and project priorities. Antenna moves currently occur every eight to ten weeks.

Fig. 1. The CARMA array at Cedar Flat in the Inyo Mountains. The control and generator buildings can be seen in the background.

2. MONITOR AND CONTROL SYSTEM ARCHITECTURE

The control system architecture is a distributed system, with intelligence embedded as close to the hardware as possible, hierarchically building up to more complex devices. This allows for object composition at several different levels, enabling code reuse. The path for monitor and control is hierarchical with little communication between components at the same layer, making dependencies very explicit.

At the lowest levels in the antennas and central electronics, embedded microprocessors (Philips C-167) are directly interfaced to the hardware. These are programmed in C by the hardware engineering staff and communicate over a 1 Mbps Controller-Area-Network bus (CANbus) as described by Woody, et al (2007)[2]. In general these microprocessors do not communicate with each other but with a CANbus master host computer that coordinates the activity of all the micros.

For the correlators a different approach is used, with custom correlator hardware boards using FPGAs with a monitor and control interface over a compact PCI (cPCI) bus. These boards are plugged into a custom cPCI chassis and

communicate over the PCI bus with a correlator master host computer in the same chassis. Each correlator baseband occupies one chassis and has one master host computer.

Both the CANbus master computers and the correlator computers are identical X86 cPCI single board computers with 1 GB of memory. These X86's are diskless and boot using the PXE protocol and are referred to as the PXE machines. There are 27 PXE machines in the 15 antenna CARMA system and another 25 in the SZA. The PXE's all run Linux (non-realtime) and communicate with the highest layer of monitor and control using Ethernet. Code in the PXE machines is in C++ and communication uses the Common Object Request Broker Architecture (CORBA) protocol. A non-realtime OS has been avoided by using queuing and sufficient compute power.

The highest level of the monitor and control system runs on a set of five server class machines in the central control building. Each of these machines is dual processor/dual core, again running a non-realtime version of Linux. These machines handle the actual execution of observing, coordinating the actions of the PXE machines in the next layer of the hierarchy. These servers also integrate the astronomical visibilities and write them to disk, handle data transfer to the archive, serve a database for monitor point values, and host user interface server programs. They are robust rack-mount machines with 6 GB of memory, dual power supplies, and RAID disks. The Array Control Computer (ACC) is the host for all the central monitor and control functions.

All of the realtime system is on a single private subnet. Additionally, several of the servers have a second interface with public Internet addresses that are visible through the firewall that are used to either serve up user interfaces to the outside world or as portals for data transfer to the archive. All of the PXE computers have 100 Mbps Ethernet, with 1 Gbps used by the servers and network switch backbones.

The monitor and control system is designed to tolerate failures in the hardware (and even the software!) as they are inevitable in a system of this size. Failed hardware components are handled using the standard methods of comparing monitor point values to limits, posting alarms, and blanking (removing) or flagging the affected data. As much as possible, an asynchronous control mechanism is used so that failed or sluggish components do not reduce the system to working at the speed of its slowest components. This sometimes requires removing failed components from communications loops so that they do not impede the flow of those that are working properly. Feedback is needed when synchronization is required with the completion of the asynchronous commands and is accomplished using the monitor system. The monitor system continuously updates the status of all monitor points every half second so that monitor point status can be used to detect command completion when necessary. All timing in the monitor and control system uses the construct of a "frame", which is one half a second, synchronized to UT. Data collection and the monitor system are synchronized to the frame.

3. MONITOR SYSTEM

The CARMA monitor system plays a central role in the overall system architecture (Amarnath, et al. 2006)[3]. While commands flow down from the control system, status information flows back asynchronously in the monitor system to the ACC. The monitor system is organized hierarchically, with the first level being the subsystem level representing a high level of abstraction, such as an antenna. There are currently 58 subsystems in the monitor system.

The monitor system is defined in XML, with an XSLT transformation into generated C++ classes that represent the full monitor system hierarchy. Base class extensions are supported as well the inclusion of common patterns. Access to any monitor point is through a specific accessor (or hierarchy of accessors), and not through a generic accessor using a string to represent a monitor point name, thus preventing runtime exceptions. An example of a hierarchical monitor point name is

Ovro3.Drive.Track.requestedAzimuth

which would be accessed by

monitorSystem.ovro(2).drive().track().requestedAzimuth()

Within a subsystem, monitor data are sent every frame from threads and processes producing monitor data to a process responsible for collecting the subsystem monitor data. These data are sent using a CORBA method call on the subsystem collection process. These subsystem processes in turn publish the subsystems data using the notification service, which are received and collated together in the ACC. Here they are made available via shared memory to any program running on this machine, which includes the central control system processes. This API to the monitor system via shared memory

includes access to a queue of recent monitor system frames and the ability to block and wait for the next monitor system frame to be available.

3.1 Monitor Points

There are approximately 55,000 monitor points in the CARMA system. The basic design philosophy is to monitor everything possible. Monitor point types are:

- Double
- Byte
- Short
- Integer
- Enum
- Boolean
- Float
- Double
- Time
- Complex
- String

Each monitor point is sampled at least once every frame and must be refreshed at that rate unless declared to be persistent. If a monitor point is not updated in a timely way then its state is declared to be invalid. This construct allows the monitor system to become a distributed continuous heartbeat for the whole system.

An XML attribute is used to assign a priority to a monitor point that can be used as a filter for database persistence. Additionally, thresholds can be defined for warning and error states. These may be over-ridden at runtime to cope with thresholds that change with time.

3.2 Monitor Point Timescales

While all monitor point data is available on the frame timescale, it is also processed on two other timescales: a one minute (1MIN) timescale, and the astronomical integration (AI) timescale. The AI monitor data is synchronized to the integrated visibilities so that monitor data can be used in astronomical data headers or directly compared with the visibilities. Both of these timescales contain the mean, maximum, and the minimum over the timescale.

3.3 Monitor Point Database

The monitor point data for the three timescales are written into flat files, with each file containing the data for a specific datatype over a limited time range. The files are then ingested into a monitor point RDBMS at Cedar Flat that is primarily used by local engineers. The 1MIN and AI timescale files are also transported to NCSA where they can be used to populate a similar database and be made available to users of the instrument. The intent is to keep the 1MIN and AI available over the life of the instrument but the frame data rolls over in about two weeks and is only available at Cedar Flat. The current database rates are about 4 TB/year.

4. CONTROL SYSTEM

4.1 Architecture and subarrays

A subarray is an essential construct for the control system, with five fixed subarrays allocated for the entire CARMA system. The first two subarrays are science subarrays, ‘sci1’ and ‘sci2’, with the first assigned the 15 station correlator and the second the eight station correlator. The system supports assignment of any antenna to a science subarray, up to the number of inputs supported by the correlator. Each science subarray has an independent reference LO. As the names imply, the science subarrays control hardware for full science support. There are two engineering arrays, ‘eng1’ and ‘eng2’, that can be used for limited engineering tests that do not require a correlator. The first engineering array has control of the spare reference LO and can thus be used for receiver or tuning tests. The second engineering array passively shares the reference LO from the first science subarray, without control of the LO. The fifth subarray is for offline antennas. The flexible assignment of the antennas is supported in the software and has been used extensively for optical pointing and testing on subsets of antennas. The full utilization of subarrays awaits completion of switches for the reference LOs and the IF assignment to the correlators.

Each of the PXE computers is used to host one to a few Distributed Objects (DOs) that provide the interfaces used by the high level control system. These interfaces are very important, both from the standpoint of the system and to allow independent software development to proceed. One of the most important interfaces is the Common Antenna API because it hides the hardware differences of the disparate antenna types in the system, allowing the control system to treat them all the same. A similar abstraction is used for the two correlators.

A subarray is controlled by a single subarray control process that has a CORBA interface (the SubarrayControl interface) to externally expose control and manage resources. The methods of this interface handle the checking of parameter ranges and fan out to antennas and correlator bands that belong to the subarray. An example of a subarray control method would be 'track', which points the antennas to a specific right ascension and declination, computes and updates delay values to the lobe rotator for interferometry control and sets delay values to the correlator digitizers. It also sets internal state so that these parameters can be periodically updated in a thread to account for the changing positions of solar system objects. Another commonly used command is 'integrate' which takes a sequence of integrations with the correlator and records them.

The management of resources and communications connections is an important responsibility of the subarray control processes. Antennas are added to and removed from the subarray, and some may be unresponsive, all of which must be managed. Method execution involves sending commands to the DO's in the PXE computers using CORBA method calls, which in turn send commands over the CANbus or PCI bus. While in theory it is possible to connect directly to an antenna DO, this would defeat the resource management functions of the subarray. All communication to the array hardware normally goes through a subarray controller.

Subarray control methods (commands by the users), are expected to be executed in less than one second, and timeouts are used to enforce this. This is accomplished by having the low level software that interfaces with the hardware respond quickly. If the subarray needs to control something that will take a long time to complete, such as moving in a calibration wheel, then a state machine variable is set and the method returns. The monitor system is used to check for completion of lengthly commands, which are termed 'procedures'. In actuality there are not many procedures, examples of which are: tracking a new source, tuning receivers, moving a calibration wheel, and integrating. The 'wait for procedure completion' is actually a blocking method in itself that can be used for synchronization. In the case of a wait for something that is antenna based, such as tracking, the wait can be specified to complete when all antennas are ready, when a specific number are ready or when all except a specific number are ready. In practice we have found that the best strategy for CARMA to deal with occasionally balky drive hardware is to wait for all antennas except two to be ready. An important aspect of this wait method is that it can be canceled at any time by an external cancel command which will cause the blocked wait command to throw a "Cancelled method' exception.

4.2 Monitor system and system state

The full monitor system is available to the subarray control processes via the shared memory access previously described. The subarray control process in turn supplies methods to user interface programs and scripts to retrieve the current value of monitor points by name. This allows external programs or scripts a view into the internal state of the system that can be very powerful.

The state of all control commands (those that change the state of the subarray) set monitor points for the parameter values. These values fully describe the commanded state of a subarray, and can be used to restore a subarray to its previous state on restart. While most of the control monitor points have been saved, the replay mechanism is still under development. In the observers currently maintain a python script that is automatically invoked when the array is restarted that restores the array to a specified state.

4.3 Deployment

The distributed control system is deployed using the CORBA Implementation Repository (IMR). The IMR is a process that runs on each machine and forks off the processes that are the DO servants. CARMA uses the Orbacus implementation of the IMR for deployment. The description of a full 'CARMA system' is contained in an XML file that has host and process names and process arguments. A script is used to process the XML file and start the IMRs on each host, and send commands that tell them about their processes. A command line program or a GUI can be used to talk to the IMR and check process state and reset or restart a process. The monitor system contains a list of all CARMA processes and their state that is kept updated by an independent process that polls the IMRs and updates the status of all processes. If a critical process is down then an alarm is triggered.

5. PROJECT DATABASE, DATA ARCHIVE AND IMAGING PIPELINE

The Project Database is an XML database recently commissioned that tracks a project from submission through time allocation, observation, grading, and archive. This provides a global tracking mechanism for all aspects of a scientific project and has many different uses. Currently it is proving to be very valuable as a tool for observers to use in scheduling the instrument. It will serve administrative functions of creating reports of observing time by institution (both allocated and actually observed). The Project Database is also the front end for archive queries, allowing data selection on its many fields. The integrated view of a CARMA scientific project from end-to-end provided by the Project Database will have uses that are not yet envisioned.

Astronomical visibilities are integrated as requested by the observing commands and written at Cedar Flat into binary files called visbricks. Header information is extracted from the AI timescale monitor point files and written into XML astroheader files. The visbricks and astroheaders are the fundamental data product of CARMA, and can in principle be used to fill data formats commonly used in radio astronomy. The visbricks and astroheaders are moved to NCSA over the Internet where they are placed in the data archive. A web based data extraction tool is available that fills the data into a Miriad format datafile on the fly that is delivered to the user.

NCSA will host the pipeline reduction system, which is based on the existing BIMA pipeline. To ensure complete success, a number of important problems for fully-automated pipeline reduction of data from mm-wavelength interferometers will have to be addressed. These include the scientific optimization of pipeline reduction, fidelity assessment of automated images, and automated flagging of bad data.

6. USER INTERFACE

The CORBA subarray control DO previously described can be used quite flexibly as the basis for a varienty of user interfaces. The primary control interface used in CARMA is an ipython interface that talks to a single subarray control DO. This interface is usually a thin layer on the subarray control DO, but further flattens the namespace by wrapping all commands into the python global namespace. While object oriented thinking is a powerful programming construct, it has not proven to be of value to the users of CARMA. This may simply reflect the CARMA operational model where the user must quickly ramp up to run the array for a week and then not return for several months. One of the most powerful aspects of python is the use of default values, which make it easy to have commands appear simple in normal use, but actually be capable of supplying significantly more functionality for the expert that is willing to read the help.

One of the most common functions for the python layer is to translate an antenna parameter into a sequence of antenna numbers for the subarray controller DO. This layer can accept either a single antenna number or a list of antenna numbers and translate them into the sequence of antenna numbers required by the DO. In many cases we use 0 as a default value that means all antennas in the subarray. The python interface can be used from the command line or from a script. Here are some command examples:

```
sci1> track("3C84", ants=4)   # carma ant#4
sci1> track("3C84", ants=[2,5,6])   # carma ants #2, #5, #6
sci1> track("3C273")   # defaults to ants=0, all antennas in the subarray
sci1> wait(TRACK, ALL)  #wait for all antennas to be on source
```

The ipython command interface allows convenient interactive control as well as script execution. During commissioning the observers provided many necessary functions quickly to get the instrument up and running by writing scripts. This also served as a testbed for prototyping. Eventually most of these developments are absorbed into the C++ subarray controller for efficiency and stability.

A script generator has been written to create the observing scripts commonly used for observing. These implement common patterns such as source/phase calibrator cycles, and passband and flux calibrators. With the script generator CARMA becomes easy to use for the general astronomical community. However, experts can always exercise the full flexibility of the instrument along with the accompanying risk.

A python TCL/TK graphical interface has also been developed for optical pointing. This GUI displays the output of the optical framegrabber in realtime and allows control of zoom, contrast, brightness, and centroiding.

Monitoring of the array is done with the CARMA Realtime Display (RTD). This program is a java interface that can be run on most computers from anywhere. Many different display pages are supported, from those that summarize the state

of a subarray to low level pages for specific hardware devices. These displays support plotting, color for out of range values, and writing values to an ascii file. The display pages are generated by server programs running on the ACC, using information from the monitor system (using the shared memory access). It is easy to generate tabular displays using a subsection of the monitor system hierarchy, but customization is possible for more complex displays. Details of the OVRO and BIMA realtime display, on which RTD is based, are given in Scott 1998[4].

7. ENGINEERING SUPPORT

Supporting the engineering staff for development and debugging of array hardware requires additional tools to those required to do astronomy. The monitor system is at the core of most of the engineering support, and exposing all the hardware with RTD pages is very useful for the engineer. Additionally the monitor point database makes it possible to track historical values of specific monitor points. For engineering work it is possible to use a 'backdoor' to get direct access to the lower level DO's, thereby exposing methods that are designed for engineering use only. References to all of these DO's, such as individual receivers, are available through the python interface. Since they bypass the usual safeguards that the subarray control process affords, they are only used for specific debugging tasks that will not interfere with observing. All of the monitor system is available through the python interface, allowing control feedback loops to be written if desired.

Test systems can also be deployed in the lab using the techniques described in the control system. Again an XML file is used to describe the deployment and all the same software can be run. Several of the realtime systems can be run in emulation mode, so it is possible to test software as well as hardware with these lab systems.

8. SOFTWARE ENGINEERING AND DEVELOPMENT METHODOLOGY

CARMA began with a rather formal development process, which is somewhat unusual in the university environment. However the requirements phase is now seen in retrospect as essential to keeping the software project on schedule. The design phase which followed was useful to bring together a distributed development team and share knowledge and techniques between team members. It also provided a learning environment for those developers unfamiliar with astronomy or interferometry.

C++ was chosen as the language of choice for CARMA because libraries supporting astronomical calculations are in C or C++ and for its efficiency. The NOVAS library is used for astronomical calculations.

Java was chosen for GUIs because of its multi-platform support, and python as a modern scripting language for observing. CORBA is used as the communication protocol, with Orbacus supplying the C++ and Java implementation while Omni-ORB is used for python.

A CVS repository is used for the CARMA code base. Keeping the build clean has been a high priority since the beginning, and Tinderbox (http://www.mozilla.org/tinderbox.html) has been used successfully for this. Tinderbox uses a web page to graphically display checkins and build status as a function of time. If the build is broken it is usually obvious from the web page who was responsible. Two different builds are monitored; the first is an incremental build with unit tests that usually completes every eight minutes, while the second is a full build with integration tests that rolls over about every two hours. The quick feedback from the incremental build has proved to be very useful.

9. SPECIFIC TECHNICAL ISSUES

In the course of developing the CARMA software there were two issues that seem fundamental and worth special mention.

9.1 C++ Exception Declarations

The most challenging aspect of building a new software system is stability, which comes down to finding the causes of program crashes and fixing them. One of the keys to this is exception handling, and in C++ this can be very tricky. If an uncaught exception is thrown in a method that is not one of those declared in its signature then unexpected() is called, which defaults to abort(), preventing valuable reporting about the source of the exception. However, if no exceptions are declared then the exception is passed to the calling method until it is caught. It is not practical to declare all the exceptions that might be thrown within a method because many library methods are used and knowing or looking up all

the possible exceptions would be time intensive. Note that this is a C++ deficiency, and is not an issue in Java, for example. The CARMA coding standards relevant to C++ exceptions are:

1. Don't declare any exceptions in the method signature if it can be avoided.
2. Derive all programs from the CARMA Program class which catches and reports any uncaught exceptions that bubble to the top of a program.
3. For methods that must have exceptions declared because they derive from methods that cannot be controlled that declare exceptions, e.g. CORBA generated stubs, enclose the entire method body in a try/catch block that catches all uncaught exceptions, using the ellipsis catch block, and rethrows them as the declared exception type.

9.2 Command Fanout

The executable code of many of the subarray control methods involves a fanout over antennas and/or correlator bands. To avoid unnecessary latency, the commands must be dispatched in parallel. Support code using functors has been developed that allows this to be done conveniently, hiding the threaded implementation that does the parallel IO. Communication to each of the CORBA DO's is encapsulated in an instance of a Handle that keeps state information for the connection to the DO. If a command times-out to a Handle then an exception is thrown so that the observing script will stop and action can be taken if desired. The Handle is taken offline and no commands are executed on the DO until monitor system information is flowing from the DO. This cleanly handles the case for an isolated power problem, a broken PXE computer, or an antenna move, and keeps commands moving as quickly as possible in the system.

ACKNOWLEDGEMENTS

A special thank you is extended to all those that have contributed to CARMA construction and operations, and in particular to the talented and dedicated members of the computing team. Funds for construction and operations have been provided by grants from the National Science Foundation and by contributions from the California Institute of Technology, the University of California, Berkeley, the University of Illinois at Urbana-Champaign, and the University of Maryland. Particular gratitude is expressed for the construction funding provided by the Kenneth and Eileen Norris Foundation, the Gordon and Betty Moore Foundation, and the Associates of the California Institute of Technology.

REFERENCES

[1] Bock, D. C.-J., Bolatto, A. D., Hawkins, D. W., Kemball, A. J., Lamb, J. W., Plambeck, R. L., Pound, M. W., Scott, S. L., Woody, D. P., Wright, M. C. H., "First Results from CARMA: The Combined Array for Millimeter-wave Research", *Proceedings of the SPIE*, 6275, 36 (2006)

[2] Woody, D. P., Wiitala, B., Scott, S. L., Lamb, J. W., Lawrence, R. P., Giovanine, C., Fredsti, S. J., Beard, A., Pryke C., Loh, M., Greer, C. H., Cartwright, J. K., Gutierrez-Kraybill, C, Bolatto, A. D., \& Muchovej, S. J. C., "Controller-area-network bus control and monitor system for a radio astronomy interferometer", Rev. Sci. Instrum., 78, 094501 (2007)

[3] Amarnath, N. S., Scott, S. L., Kraybill, J. C., Beard, A. D., Daniel, P., Gwon, C., Hobbs, R., Leitch, E., Mehringer, D., Plante, R., Pound, M. W., Rauch, K. P., Teuben, P. J. "The CARMA Monitor System (CAM) – Transforming Cyclically Collected Telemetry into a Linear Stream", ASP Conf. Ser., 314, 720 (2004)

[4] Scott, Stephen, "Remote observing with the Caltech millimeter wave array", *Proceedings of the SPIE*, 3351, 118 (1998)